Departamento de Informática
Licenciatura em Engenharia Informática
8 de Maio de 2015

Universidade do Minho
Escola de Engenharia

Programação Lógica

Trabalho Prático 2

Sistemas de Representação de Conhecimento e Raciocínio

Grupo 15

Orlando José Gomes Martins Costa a67705
Paulo Ricardo Cunha Correia Araújo a58925
Rui Pedro Azevedo Oliveira a67661

Resumo

O presente documento relata o desenvolvimento de um sistema de representação de conhecimento e raciocínio com a capacidade para caracterizar um universo de comércio automóvel, e serve de suporte ao trabalho realizado.

Este documento encontra-se estruturado sob a forma de um relatório de desenvolvimento técnico, e reflete o trabalho realizado ao longo da 2ºfase da componente prática da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Numa fase inicial do relatório são apresentados os predicados desenvolvidos de forma a retratar o sistema, assim como as estratégias de desenvolvimento que modelaram a solução. Tendo definida a base sob a qual trabalhar, são então descritos os mecanismos de representação de conhecimento perfeito e imperfeito idealizados, assim como os invariantes que restringem a inserção e remoção de conhecimento contido na base de conhecimento.

Finalmente, encontram-se retratados os procedimentos responsáveis pela evolução do conhecimento, assim como o sistema de inferência desenvolvido que permite traduzir os mecanismos de raciocínio inerentes ao sistema.

As ferramentas utilizadas recaem sobre a utilização de uma extensão à programação em lógica, usando a linguagem de programação PROLOG, assim como uma “camada” de interação com a base lógica em linguagem Java, com o auxílio da biblioteca *Jasper*.

Conteúdo

1 Introdução

O propósito deste trabalho reside no desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de comércio automóvel. Para isso foi desenvolvido um sistema baseado nos requisitos funcionais necessários à solução das 5 questões propostas, apresentadas seguidamente. Isto é alcançado através da inferência de predicados lógicos e invariantes numa base de conhecimento, correspondentes a diversos relacionamentos entre um dado identificador e um dado parâmetro, como p.e. marca, modelo ou fabricante. O sistema de inferência é alcançado através da simulação de operações lógicas de conjunção e disjunção, capaz de obter o resultado lógico de operações compostas com vários predicados.

De forma a possibilitar a demonstração de representação de conhecimento positivo e negativo, é necessária a criação de predicados que tornem possível a evolução e retrocesso de conhecimento, assim como a utilização de invariantes com o intuito de não possibilitar a representação de situações impossíveis no mundo real.

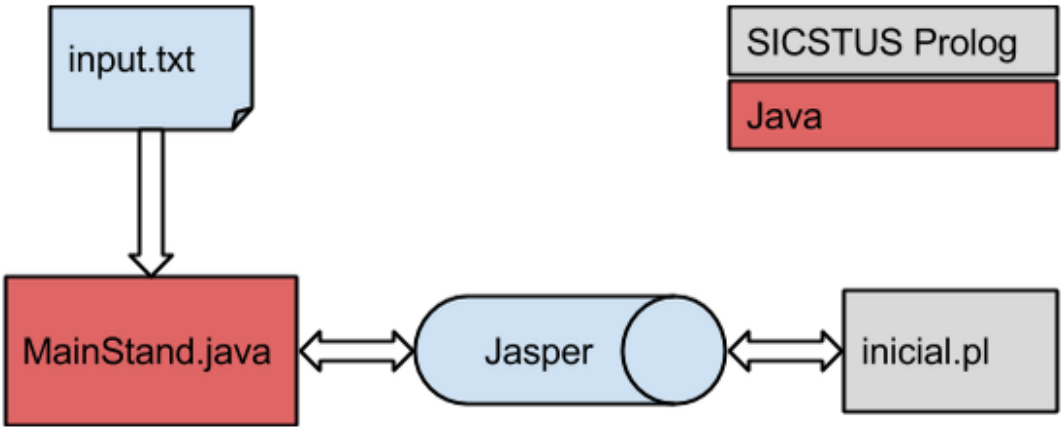
O conhecimento imperfeito deve também ser representado explorando o princípio de mundo aberto (PMA), estando todos os seus diferentes tipos definidos através do relacionamento entre predicados ou a negação destes.

Para lidar melhor com os vários tipos de conhecimento a evolução da base de conhecimento deverá ser adaptada, de forma a poder-se evoluir de conhecimento imperfeito para conhecimento perfeito.

O objetivo final do sistema é então possuir a capacidade de representar diferentes tipos de conhecimento assim como resolver problemas através da construção de mecanismos de raciocínio e de inferência.

2 Arquitetura do Sistema

Um dos requisitos do exercício consiste em criar uma aplicação em Java capaz de interagir com SICStus Prolog. Assim, foi desenvolvida utilizando a linguagem java, uma "camada" superior à base de conhecimento criada em Prolog, com a qual o utilizador pode facilmente evoluir esta base e efetuar perguntas lógicas no domínio do mundo aberto.



A conexão entre estas duas linguagens é efetuada através de uma biblioteca denominada Jasper. Jasper funciona como uma interface bi-direcional entre estes dois tipos de programas. O lado da interface integrado em java consiste num conjunto de classes (package) representativas do emulador SICStus. Este biblioteca permite então mapear os métodos em java nos predicados Prolog, convertendo os argumentos.

2.1 Extras do sistema

Ao longo do desenvolvimento do sistema foram utilizados alguns componentes extra que serviram de auxílio na depuração de erros. Assim, além dos comandos referidos, o sistema é capaz de:

- interpretar comentários no ficheiro input.txt (comando $$ <comentario>)
- executar diretamente prolog do ficheiro input.txt (comando ## <progog>)
- detetar qual o invariante que falhou numa evolução ou retrocesso (predicado whyevo e whyrem)

3 Predicados

Os predicados criados para complementar a base de conhecimento do sistema são:

- marca (relaciona uma dada matrícula/id com uma determinada marca automóvel)
- modelo (relaciona uma dada matrícula/id com um determinado modelo automóvel)
- fabricante (relaciona um dado modelo automóvel com uma determinada marca automóvel)
- proprietário (relaciona uma dada matrícula/id com um determinado nome de proprietário)

Existe a possibilidade de adicionar novos tipos de predicados à base de conhecimento, sendo que como exemplo e para manter a simplicidade na demonstração, o grupo decidiu apenas implementar 4 predicados, decisão aprovada pelo professor.

4 Representação de conhecimento

Durante este exercício é explorado o principio do mundo aberto (PMA). Este, contrariamente ao princípio do mundo fechado (PMF), assume que o valor de algo que não se sabe ser "verdade" é simplesmente "desconhecido", enquanto que no exercício anterior, foi assumido que algo que não se sabe ser verdade deve ser falso.

Este novo princípio aplica-se quando um sistema possui informação completa. É então utilizado quando queremos representar conhecimento (ontologias) assim como descobrir nova informação. Por exemplo, se o historial clinico de um paciente não referir uma alergia em particular, seria incorreto afirmar que esse paciente não sofre dessa alergia.

4.1 Evolução de Conhecimento

Como abordado nas aulas práticas, a evolução de conhecimento foi realizada através do predicado 'evolucao(T)', sendo que este utiliza o predicado 'solucoes (I,+T::I,S)', que trata de encontrar todos os invariantes relacionados com a inserção de conhecimento de 'T', assim como o predicado 'insere(T)', que trata de inserir 'T' na base de conhecimento e por fim o predicado 'teste(S)' que, efetuada a inserção, trata de testar todos os invariantes encontrados anteriormente. Caso alguma verificação de um invariante falhe, não é feita a evolução.

Evolução permite inserir o conhecimento requerido respeitando os invariantes definidos e mantendo a base de conhecimento consistente.

4.1.1 Positivo

Invocação:

No interpretador, para adicionar nova informação positiva, utiliza-se o seguinte comando:

evolucaoPos <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende adicionar a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha.

Exemplo de input:

evolucaoPos modelo '40-NT-69' 'A5'

Prolog gerado:

evolucao(modelo('40-NT-69','A5')).

4.1.2 Negativo

Invocação:

No interpretador, para adicionar nova informação negativa, utiliza-se o seguinte comando:

evolucaoNeg <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende adicionar a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha.

Exemplo de input:

evolucaoNeg proprietario '91-FF-32' 'Orlando Cunha'

Prolog gerado:

evolucao(-proprietario('91-FF-32','Orlando Cunha')).

4.2 Retrocesso de conhecimento

A remoção de conhecimento foi realizada através do predicado 'retrocesso(T)' que utiliza o predicado 'solucoes(I,-T::I,S)', que trata de encontrar todos os invariantes relacionados com a remoção de conhecimento de 'T', o predicado 'apaga(T)', que trata de remover 'T' na base de conhecimento e o predicado 'teste(S)', que, efetuada a remoção, trata de testar todos os invariantes encontrados anteriormente. Caso alguma verificação de um invariante falhe, não é feito o retrocesso.

O retrocesso permite remover conhecimento requerido respeitando os invariantes definidos e mantendo a base de conhecimento consistente.

4.2.1 Positivo

Invocação:

No interpretador para remover informação positiva, utiliza-se o seguinte comando:

retrocessoPos <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende remover a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha.

Exemplo de input:

retrocessoPos marca '90-MG-60' 'Mitsubishi'

Prolog gerado:

retrocesso(marca('90-MG-60','Mitsubishi')).

4.2.2 Negativo

Invocação:

No interpretador para remover informação negativa, utiliza-se o seguinte comando:

retrocessoNeg <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende remover a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha.

Exemplo de input:

retrocessoNeg modelo '49-VM-94' 'Patrol'

Prolog gerado:

retrocesso(-modelo('49-VM-94','Patrol')).

5 Representação de conhecimento imperfeito

Os tipos de conhecimento imperfeito a ser implementado são: incerto, impreciso e interdito.

5.1 Conhecimento Incerto

O conhecimento incerto é utilizado quando um dado predicado é possivelmente verdade, mas não se sabe ao certo quais são os parâmetros que o tornam verdade.

O conhecimento incerto é inserido através da utilização de um símbolo reservado para a informação que é incerta. Este símbolo é será único por registo de forma a ser possível separar os diferentes casos de conhecimento incerto.

A exceção, que permitirá ao demonstrador (demo) obter o resultado de desconhecido, é derivada utilizando uma implicação que usa o símbolo reservado.

(Exemplo: 'excecao(proprietario(A,B)) :- proprietario(A, *incerto01*)).').

Invocação:

No interpretador para adicionar nova informação incerta, utiliza-se o seguinte comando:

evolucaoIncerto <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende adicionar a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha, notando-se que o argumento que é incerto terá necessariamente que ter o valor 'incerto'.

Exemplo de input:

evolucaoIncerto proprietario *'10-FA-32' incerto*

Prolog gerado:

*evolucao(*proprietario*('10-FA-32',incerto01))*

assert((excecao(proprietario(A,B)) :- proprietario(A, *incerto01*))).

Onde incerto01 é único.

5.2 Conhecimento Impreciso

O conhecimento impreciso é utilizado quando existem certos parâmetros que podem tornar o predicado verdadeiro, pertencendo estes a um conjunto conhecido. No entanto, não é possível saber se tornam ou não o predicado verdadeiro.

O conhecimento impreciso é guardado na forma de exceção. Desta forma o demonstrador irá gerar dar como resultado desconhecido quando se questionar sobre algum conhecimento que seja impreciso.

Invocação:

No interpretador para adicionar nova informação imprecisa, utiliza-se o seguinte comando:

evolucaoImpreciso <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende adicionar a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha.

Exemplo de input:

evolucaoImpreciso proprietario '10-FA-33' 'Rui Oliveira'

Prolog gerado:

evolucao(excecao(proprietario('10-FA-32','Rui Oliveira')))

5.3 Conhecimento Interdito

O conhecimento interdito utilizando quando se quer representar conhecimento que não pode ser acedido.

O conhecimento interdito é guardado de forma equivalente ao incerto, utilizando-se um símbolo reservado para a informação que é interdita, símbolo que também será único por registo.

Para marcar este símbolo como reservado existe o predicado 'interdito(A)'.

Invocação:

No interpretador para adicionar nova informação incerta, utiliza-se o seguinte comando:

evolucaoInterdito <predicado> <argumentos>

Legenda:

<predicado> nome do predicado sobre o qual se pretende adicionar a informação.

<argumentos> lista de argumentos que se pretende que o predicado tenha, notando-se que o argumento que é interdito terá necessariamente que ter o valor 'interdito'.

Exemplo de input:

evolucaoInterdito proprietario '10-FA-32' interdito

Prolog gerado:

evolucao(proprietario('10-FA-32',interdito1))

evolucao(interdito(interdito1))

Onde interdito1 é único.

6 Invariantes para restrições à inserção e remoção

6.1 Análise

Foi necessária a criação de um invariante que não permitisse a inserção de conhecimento repetido, ou seja, caso exista na base de conhecimento uma relação marca(P,B), esta relação não pode voltar a ser inserida.

6.1.1 Implementação

Na implementação deste invariante, começamos por colocar todas as relações marca(P,B) numa lista S. De seguida, garantimos que existe apenas um elemento nessa lista.

Para as relações modelo(P,M), fabricante(B,M) e proprietario(P,O) também foram criados invariantes do mesmo género.

```
+marca( P,B ) :: (solucoes( (P,B),(marca( P,B )),S ),
                  comprimento( S,N ),
                  N == 1
                  ).
```

6.2 Análise

Foi criado um invariante que não permite que um modelo tenha mais que uma marca associada. Sendo assim, existindo a relação fabricante(B,M), não é que a relação fabricante(C,M) seja inserida.

6.2.1 Implementação

Na implementação deste invariante, começamos por colocar todas as marcas associadas ao modelo que pretendemos inserir numa lista S. De seguida, garantimos que apenas existe um elemento nessa lista, ou seja, apenas uma marca pode fabricar um determinado modelo.

```
+fabricante( B,M ) :: (solucoes( X,(fabricante( X,M )),S ),
                       comprimento( S,N ),
                       N == 1
                      ).
```

6.3 Análise

De forma a impedir que uma matrícula estivesse associada a vários(as) proprietários/marcas/modelos, foi criado um invariante para esse tipo de situações.

6.3.1 Implementação

Na implementação deste invariante, começamos por encontrar e colocar todos os proprietários associados à matrícula a inserir numa lista S. De seguida, garantimos que apenas existe um elemento nessa lista, ou seja, apenas uma pessoa é proprietária da respetiva matrícula.

Para as relações modelo(P,M) e marca(P,M) também foram criados invariantes do mesmo género.

```
+proprietario( P,O ) :: (solucoes( M,(proprietario( P,M )),S ),
            comprimento( S,N ),
            N == 1
            ).
```

6.4 Análise

De forma a impedir situações em que uma matrícula é registada num modelo que não pertence à marca que esta foi registada, foi necessária a criação de um invariante.

6.4.1 Implementação

Na implementação deste invariante, começamos por encontrar todas as marcas que pertencem à interseção entre a marca associada à respetiva matrícula e a marca que fabrica o respetivo modelo, e colocamos numa lista S. De seguida, indicamos que essa lista apenas pode ter um elemento, ou seja, a marca que a matrícula está associada e a marca que fabrica o modelo associado à mesma matrícula, são

a mesma. Como alternativa, também podemos aceitar que o modelo a que pretendemos associar uma matrícula, ainda não se encontre registado, sendo assim, começamos por encontrar todas as marcas que fabricam o respetivo modelo e, caso não exista nenhuma, então aceitamos a inserção na base de conhecimento.

```
+modelo( P,M ) :: ((solucoes( B,(marca( P,B ), fabricante( B,M )), S ),
                comprimento( S,N ),
                N == 1
                ) ; (
                solucoes( B1,(fabricante( B1,M )), S1 ),
                comprimento( S1,N1 ),
                N1 == 0
                )).
```

6.5 Análise

Foi necessário um invariante que restringisse a remoção de modelos quando ainda existissem matrículas a eles associadas. Por exemplo, caso exista uma matrícula 'P' associada a um modelo 'M', não deve ser possível a remoção deste modelo M da base de conhecimento.

6.5.1 Implementação

Na implementação deste invariante, é necessário verificar se não existem matrículas associadas ao modelo que pretendemos remover da base de conhecimento. Sendo assim, começamos por encontrar todas as matrículas P associadas ao modelo M, que pretendemos remover, e colocamos numa lista S. Por fim, verificamos se essa lista S está vazia, ou seja, não existem matrículas associadas ao modelo em questão.

```
-fabricante( B,M ) :: (solucoes( P,(modelo( P,M )), S),
                comprimento( S,N ),
                N == 0
                ).
```

6.6 Análise

Por último, foram necessários invariantes de forma a impedir situações em que se pretende inserir conhecimento negativo, sendo que já exista conhecimento positivo para um determinado caso, ou seja, caso pretendamos representar que a marca “Renault” fabrica o modelo “Fiesta”, mas já exista na base de conhecimento informação que represente o contrário, então esta inserção não deverá ser possível.

6.6.1 Implementação

Em primeiro lugar começamos por verificar se existe conhecimento negativo para a informação que está a ser inserida. Não deve ser possível esta inserção de conhecimento, caso exista conhecimento negativo. O mesmo se aplica para todos os outros predicados.

```
+fabricante( P,M ) :: (solucoes( (P,M), (-fabricante( P,M )),S ),
                comprimento( S,N ),
                N == 0
                ).


+(-fabricante( P,M )) :: (solucoes( (P,M), (fabricante( P,M )),S ),
                comprimento( S,N ),
                N == 0
                ).
```

7 Mudanças no tipo de Conhecimento através de Evolução

Para situações em que existe um dado caso de conhecimento imperfeito na base de conhecimento e de seguida pretendemos inserir conhecimento perfeito, seguimos uma abordagem que permite este novo conhecimento substituir o conhecimento imperfeito prévio, dado que este possui um menor valor lógico.

De forma a lidar com a situação mencionada, é necessário que o predicado evolução seja capaz de, antes da inserir novo conhecimento, elimine o conhecimento imperfeito presenta na base de conhecimento. Desta forma, desenvolveu-se o predicado ***rmExcecao(Q)*** que recebe um predicado como parâmetro e apaga toda o conhecimento imperfeito relacionado com este.

Deste forma, é apenas necessário uma execução do predicado evolução anteriormente definido após a eliminação do conhecimento não necessário.

7.1.1 Predicado evolução

```
evolucao(Q) :-
        rmExcecao(Q),
        evolucaoNormal(Q).
```

7.1.2 Predicado rmExcecao

```
rmExcecao(Q) :-
        nao(interdito(Q)),
        solucoes((excecao(Q),A),(clause(excecao(Q),A),(A)),S),
        retractAll(S).
```

7.1.3 Predicado retractAll

```
retractAll([(P,Q)|T]) :-
        Q \== true,
        retract((P:-Q)),
        retract(Q),
        retractAll(T).
retractAll([(P,Q)|T]) :-
        retract((P:-Q)),
        retractAll(T).
retractAll([]).
```

8 Sistema de inferência

O sistema de inferência é implementado através do predicado demo, podendo tomar 3 valores lógicos (verdadeiro, falso e desconhecido).

8.1 Operadores

De forma a facilitar a leitura dos diversos predicados demo implementados, foram definidos dois simbolos de operação "e" e "ou", que correspondem respetivamente à conjunção e disjunção de valores. Foram definidos com o mesmo valor de precedência e de aparecimento de parâmetros que os dois símbolos originais(\/ e /\ ), e são utilizados no predicado demo entre ambos os predicados a serem avaliados.

```
:- op( 500,yfx,'e' ).

:- op( 500,yfx,'ou' ).
```

O predicado demo possui várias cláusulas, que resolvem todos os casos de operação lógica correspondentes. Foram implementados seguindo a metodologia apresentada nas seguintes tabelas:

INPUT		OUTPUT
A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

Tabela 1 - Disjunção

INPUT		OUTPUT
A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

Tabela 2 - Conjunção

Para obter o valor lógico de um predicado simples é utilizado o predicado demo definido nas aulas, sendo este valor utilizado então nos predicados mais complexos definidos como:

- A V B com A->F e B->D

demo((A ou B), desconhecido) :- demo(A, falso), demo(B, desconhecido).

8.2 Lógica do predicado demo

- um dado predicado apenas é verdadeiro se existe confirmaçao positiva que esse predicado é verdadeiro;
- um dado predicado é falso apenas se existe confirmaçao que esse predicado é falso;
- caso nao existe confirmaçao positiva nem negativa, o valor lógico desse predicado é desconhecido;
- o valor das conjunçoes e disjunçoes dos predicados foram tidos em conta com o valor lógico que possuem.

Nota: Um predicado é falso quando não existe confirmação que este é verdadeiro nem a existência do predicado exceção relacionado com este. Estes predicados relacionam-se para o valor de um predicado quando este é negativo.

```
-fabricante( Fabricante,Marca ) :-
        nao(fabricante( Fabricante,Marca )),
        nao(excecao( fabricante( Fabricante,Marca ))).
```

Os valores que o mecanismo assume dependem do "operação lógica" realizada. Nos predicados demo de conjunção, é tomado sempre o valor mais fraco dos predicados em causa (entenda-se mais fraco como verdadeiro > desconhecido > falso). Nos predicados demo de disjunção, contrariamente à conjunção, é tomado sempre o valor mais forte.

É então possível através do encadeamento de predicados demo, obter o valor lógico de um conjunto de predicados, variável consoante o uso de conjunção e/ou disjunção.

9 Testes Realizados

De forma a validar o trabalho realizado, foi efetuada uma série de testes. Os testes efetuados, assim como o resultado esperado e o resultado obtido são apresentados de seguida:

Nota:
OE - Output Esperado, **OO** - Output Obtido

Precedente

- evolucaoPos modelo '43-OF-31' 'C5'
- evolucaoPos marca '22-RO-06' 'Alfa Romeo'

Input	OE	OO
demo (modelo('43-OF-31','C5') ou marca('22-RO-06','Alfa Romeo'))	V	V

Precedente

- evolucaoPos modelo '43-OF-31' 'C5'
- evolucaoNeg marca '91-FF-32' 'BMW'

Input	OE	OO
demo (modelo('43-OF-31','C5') ou marca('91-FF-32','BMW'))	V	V

Precedente

- evolucaoNeg marca '91-FF-32' 'BMW'
- evolucaoPos modelo '43-OF-31' 'C5'

Input	OE	OO
demo (marca('91-FF-32','BMW') ou modelo('43-OF-31','C5'))	V	V

Precedente

- evolucaoNeg marca '91-FF-32' 'BMW'
- evolucaoNeg modelo '91-FF-32' 'Ducato'

Input	OE	OO
demo (marca('91-FF-32','BMW') ou modelo('91-FF-32','Ducato'))	F	F

Precedente

- evolucaoNeg modelo '91-FF-32' 'Ducato'
- evolucaoIncerto modelo '10-WW-99' incerto

Input	OE	OO
demo (modelo('91-FF-32','Ducato') ou modelo('10-WW-99', 'Ferrari'))	D	D

Precedente

- evolucaoImpreciso modelo '49-VP-94' 'Patrol'
- evolucaoNeg modelo '91-FF-32' 'Ducato'

Input	OE	OO
demo (modelo('49-VP-94','Patrol') ou modelo('91-FF-32','Ducato'))	D	D

Precedente

- evolucaoPos marca '22-RO-06' 'Alfa Romeo'

Input	OE	OO
demo (marca('22-RO-06','Alfa Romeo') ou marca('91-JH-34','BMW'))	V	V

Precedente

- evolucaoPos marca '22-RO-06' 'Alfa Romeo'

Input	OE	OO
demo (modelo('91-GF-02','Ducato') ou marca('22-RO-06','Alfa Romeo'))	V	V

Precedente

- evolucaoPos marca '22-RO-06' 'Alfa Romeo'
- evolucaoPos proprietario '43-OF-31' 'Fernando Gomes'

Input	OE	OO
demo (marca('22-RO-06','Alfa Romeo') e proprietario('43-OF-31','Fernando Gomes'))	V	V

Precedente

- evolucaoNeg marca '91-FF-32' 'BMW'
- evolucaoNeg modelo '91-FF-32' 'Ducato'

Input	OE	OO
demo (marca('91-FF-32','BMW') e modelo('91-FF-32','Ducato'))	F	F

Precedente

- evolucaoPos marca '22-RO-06' 'Alfa Romeo'
- evolucaoNeg marca '91-FF-32' 'BMW'

Input	OE	OO
demo (marca('22-RO-06','Alfa Romeo') e marca('91-FF-32','BMW'))	F	F

Precedente

- evolucaoNeg marca '91-FF-32' 'BMW'
- evolucaoPos marca '22-RO-06' 'Alfa Romeo'

Input	OE	OO
demo (marca('91-FF-32','BMW') e marca('22-RO-06','Alfa Romeo'))	F	F

Precedente

- evolucaoPos marca '40-NT-69' 'Audi'
- evolucaoImpreciso modelo '34-PP-23' 'Série 5'

Input	OE	OO
demo (marca('40-NT-69','Audi') e modelo('34-PP-23','Série 5'))	D	D

Precedente

- evolucaoImpreciso modelo '34-PP-23' 'Série 5'
- evolucaoPos marca '40-NT-69' 'Audi'

Input	OE	OO
demo (modelo('34-PP-23','Série 5') e marca('40-NT-69','Audi'))	D	D

Precedente

- evolucaoNeg modelo '91-FF-32' 'Ducato'

Input	OE	OO
demo (modelo('91-FF-32','Ducato') e marca('91-JH-34','BMW'))	F	F

Precedente

- evolucaoNeg modelo '91-FF-32' 'Ducato'

Input	OE	OO
demo (marca('91-JH-34','BMW') e modelo('91-FF-32','Ducato'))	F	F

10Conclusão e Sugestões

Durante a realização do trabalho, foram desenvolvidos alguns predicados que permitem validar o sistema de inferência criado, assim como diversos invariantes que reforçam a consistência destes predicados. Apenas foram desenvolvidos 4 predicados por forma a simplificar o conteúdo da base de conhecimento, sendo estes os adequados para demonstrar a capacidade de inferência do sistema. Os predicados relacionam-se logicamente entre si, sendo óbvio o relacionamento entre os predicados "marca" -> "fabricante" -> "modelo". Os predicados "proprietário", "marca" e "modelo" utilizam como identificador universal a matrícula do automóvel em causa, tornando desnecessária a inserção de um parâmetro extra nestes. O sistema de inferência criado é capaz de derivar respostas partindo de conhecimento imperfeito e aceita um número indefinido operações lógicas para o fazer. A consistência da base de conhecimento foi assegurada através de diversos invariantes que garantem coerência nas respostas às perguntas efetuadas.

A camada criada em java que interage com a base de conhecimento em Prolog permite ao utilizador uma utilização simples, e possui diversas funcionalidades essenciais e extra que tornam a experiência de utilização relativamente fácil.

Durante a modelação do exercício, deparamo-nos com alguns problemas que nos obrigaram a tomar diferentes caminhos da abordagem inicial. Esta abordagem consistia numa base de conhecimento demasiado ambiciosa e dependente de demasiados predicados, pelo que nos focamos num conjunto de predicados menor que cumpre todos os requisitos propostos. Existiu também alguma problemática em volta do predicado evolução, pois implementar o predicado que apaga o conhecimento imperfeito exigiu alguma criatividade. Ainda assim o predicado rmExcecoes não está perfeito e possui algumas falhas, como por exemplo em situações nas quais o predicado evolução antigo falha, o rmExcecoes não consegue recuperar o conhecimento imperfeito antigo.

Em forma de crítica final, apesar dos problemas encontrados, o grupo sente-se confiante na sua habilidade de cumprir os desafios propostos e crê ter atingido o objetivo subjacente na realização do trabalho, tendo criado um sistema de representação de conhecimento e raciocínio simples com a capacidade de representar vários tipos de conhecimento assim como inferir o valor lógico de um conjunto de predicados.